Networked Traffic Control Smart City Project

1.Divya Bharti Sharma
Faculty of Computer Science
Fachhochschule Dortmund – University of Applied Sciences and Arts

2.Jahnavi Vennapusa
Faculty of Computer Science
Fachhochschule Dortmund – University of Applied Sciences and Arts

3.Sebastian Rudolph
Faculty of Computer Science
Fachhochschule Dortmund – University of Applied Sciences and Arts

4.Vasanth Kandiyathevan
Faculty of Computer Science
Fachhochschule Dortmund – University of Applied Sciences and Arts

***Abstract*—This paper is the final submission for the smart city semester project of the embedded systems engineering lecture. It includes the development process of the use case of a smart parking guidance system from the requirements over design and implementation to testing.**

***Index Terms*—smart city, parking guidance, development, real time simulation,testing,System schedulable**

I. Introduction

There are various technologies spread throughout a smart city, such as Edge Computing, Internet of Things (IoT), artificial intelligence (AI),sensors, and so on also real time operating system which used to collect and analyze data about the city and its residents, as well as to make daily life easier. The data is then used to make informed decisions and implement solutions to various city challenges, such as traffic management, energy consumption, waste management, agriculture, and so on.By optimizing traffic flow, improving public transportation, and encouraging alternative modes of transportation, smart city technology has the potential to significantly improve traffic control in urban areas. This can result in less congestion, better air quality, and a higher quality of life for citizens.

II. Short overview

A. Use-case description

Since the broader project-context of a smart city in general is far too complex and multilayered to be covered in the frame of a semester project, we decided to focus our work on the use-case of a smart parking guidance system. Such a parking guidance represents a interconnected system that depends on multiple components within the eco-system of a smart city. With this, we found a good balance between simplicity on the one hand and interaction between different components on the other hand. We started our work by defining ten basic requirements for the smart city in general. This requirements evolved a bit during the development process since we found some ideas being not feasible.
Our final requirements to a smart city where:

1) V2I (Vehicle to Infrastructure communication)
2) Pollution monitoring
3) Smart parking guidance
4) Adaptive traffic control
5) Integrated emergency system
6) Waste management
7) Sustainable energy management
8) Safety
9) Autonomous delivery robots
10) Noise monitoring

Some of the mentioned requirements are only theoretical, since not all of them could be realised in our proof of concept implementation. Others were alternated, as for example no real cloud service or C2I interface was available.

B. Characteristics of the application domain

1) Automotive Domain: Below mentioned different Characteristics of embedded system use in automotive industries are

a)Multifunctional Camera:It is an Electronic control unit which has a sensor(camera with lens on it) with microcontroller unit with either 32 or 64 flash memory also it include cable connection to connect with higher level of bus networks (such as CAN/LAN/Flex ray).The multipurpose camera for assisted and partially automated driving utilizes an innovative, high-performance system-on-chip (SoC)

b)Reactive system:Embedded Communications The increasing complexity of electronic architectures embedded in a vehicle, and locality constraints for sensors and actuators,

has led the automotive industry to adopt a distributed approach for implementing the set of functions. In this context, networks and protocols are of primary importance.

c)Real time system Power Train: The power train and chassis domains both exhibit hard real-time systems and a need for high computation power. The Power train domain represents the system that controls the engine according to requests from the driver (for example: speeding up, slowing down as transmitted by the position sensor or the brake pedal, etc.) and requirements from other parts of the embedded system such as climate control.

d)Distributed system: Chassis Domain: It is a system which basically aims to control the interaction of car with road such as suspension wheel etc. whereas controllers consider the requests emitted by driver for steering, braking. This domain includes ABS (Anti-lock braking system).

e)Dependable system: Human- Machine Interference: It is an interaction between driver and the numerous embedded function which is in the vehicle. The major functionalities are about the status of car, vehicle speed.

f)Security: It can be computer security which aware about cyber risks related to all autonomous driving protect from harmful attacks.

g)Dependability: The main features include the dependability of Powertrain, control system, and seats.

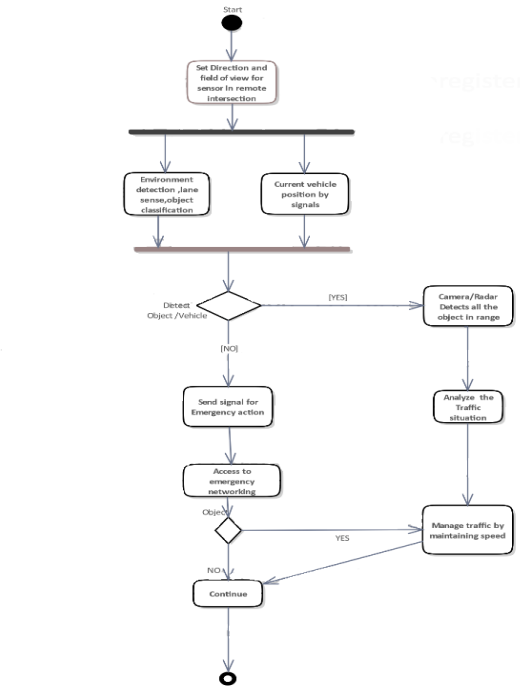

Fig. 1. Activity diagram for a traffic control in remote place

C. Activity Diagrams

We have referred to two three activity diagram references such as activity at remote areas also parking system and emergency situation.

1) Activity diagram of obstacle detection at remote areas: The main Concern here shown in below figure as we can observe any obstacle or any latency can be detected by sensor which are being placed on the vehicle. If any obstacle being measured sensor needs to send the information to traffic management within milliseconds the action has to be taken either by maintaining speed of the vehicle or changing the direction of vehicle as shown in fig 1.

2) Activity diagram of Smart Parking System: The smart parking approach we have considered is to reduce the user burden of finding a parking slot as they may need to search the entire parking space looking for empty slot. This approach assists the user in finding the parking slot in most simplest way as shown in fig 2.

3) Activity diagram of traffic control: Finally we created an activity diagram for a adaptive traffic control system in a smart city environment. The traffic control system is supposed to control the traffic flow in the city in a way, so that areas with high traffic volume are relieved. The system must also

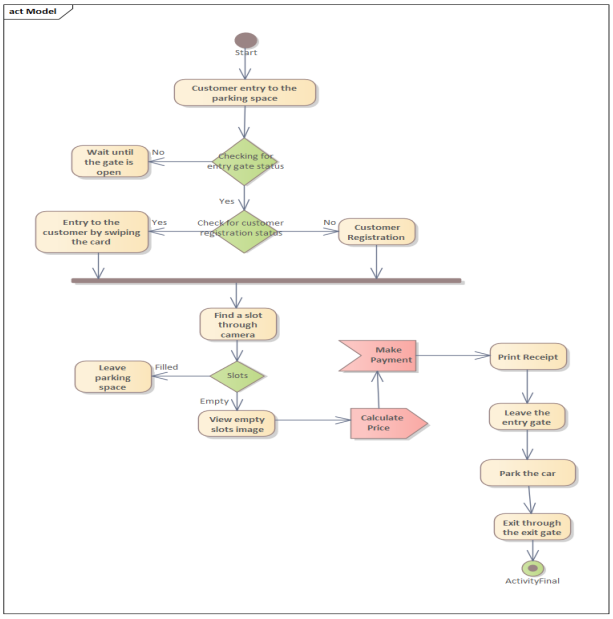

Fig. 2. Activity diagram for smart parking

give a free way for emergency services and be connected to the air pollution system, so that highly polluted areas are also relieved from higher traffic volume to lower the air pollution.

III. Design Analysis

Here the implementation of all the requirements which is being considered in Smart city we have filter out a analysis of parking system for traffic control in road while parking the vehicle.

A. 1.Block Diagram

We have implemented block diagram by the help of different blocks in enterprise architect tool whereas main focus on a smart city outcome by providing different requirements such as parking system, vehicle to infrastructure also reducing time effort for humans apart from automated vehicle it can be robots, drones for day to day life as shown in fig.3

B. 2.Internal Block Diagram

The implementation of two particular requirements has considered where one block such as parking block from block definition diagram defined in a way as shown figure 4.

Here the main system revolves around parking system. The second block we have taken such as weather monitoring system also shown in Figure 5

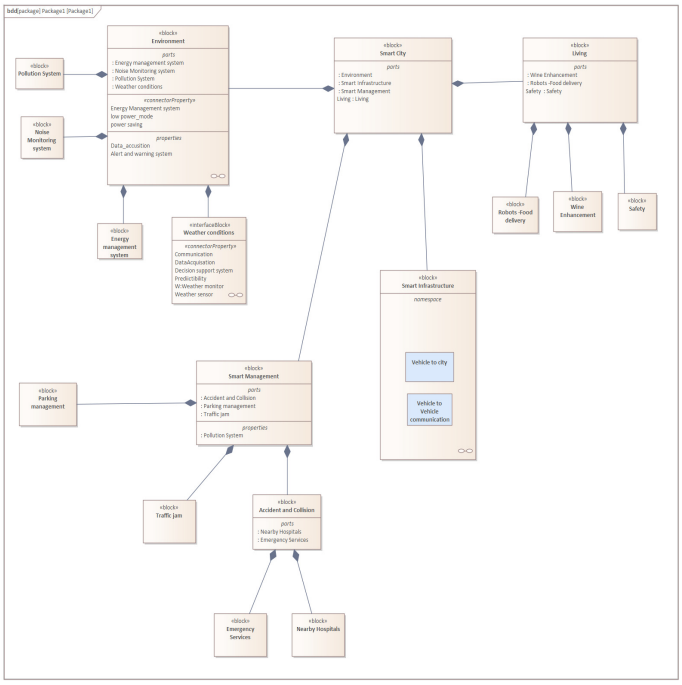

Fig. 3. Block Definition Diagram

C. 3.Parametric Diagram

The need for diagram is to provide necessary parameters also analysing them with the help of mathematics which we refer as equations. As we focused on a traffic control for the parking system here we have given different parameters for vehicle to be maintained for parking management are in below mentioned figure 6

1. Time 2. Alarm 3. Distance 4. Height. 5. Sensor frequency

To provide mathematical perspective we have used are described in figure 7

"Parametric Constrained Diagram"

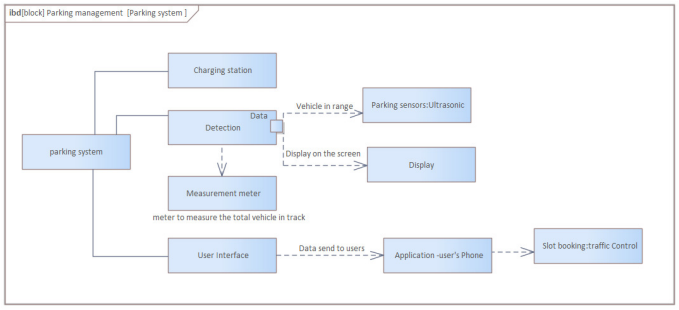

Fig. 4. Internal block of Parking requirement

D. 4.Sequence Diagram

To illustrate the order of processes for different use-cases in a smart city, we created a sequence diagram. In this diagram, a citizen or its car - depending on the use-case - take up the role of the actor, while the infrastructure with its sensors and the cloud computing take up two layers in the system.

E. State Machine implementation for traffic control system

Before we analyse our use case from the previous reference we have discussed and get a result of around two different state machines which are being related to parking system

1) a.State Machine of parking system: In this case we have considered the states which a vehicle can transit while parking. As shown in the state machine diagram vehicle will be in idle state.The next state would be checking for parking slots available state. Further ahead vehicle will be transiting to parking state or exit state based on slot availability.

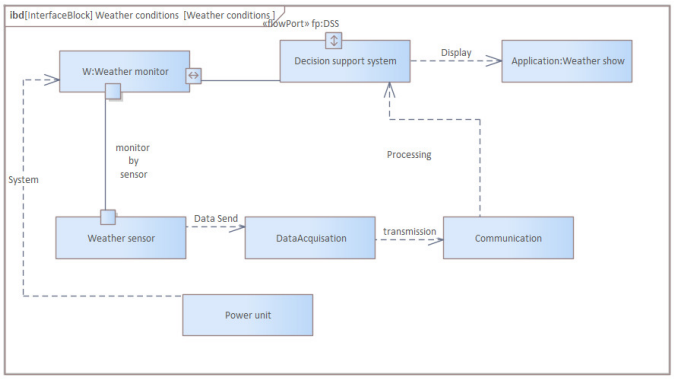

Fig. 5. Weather system

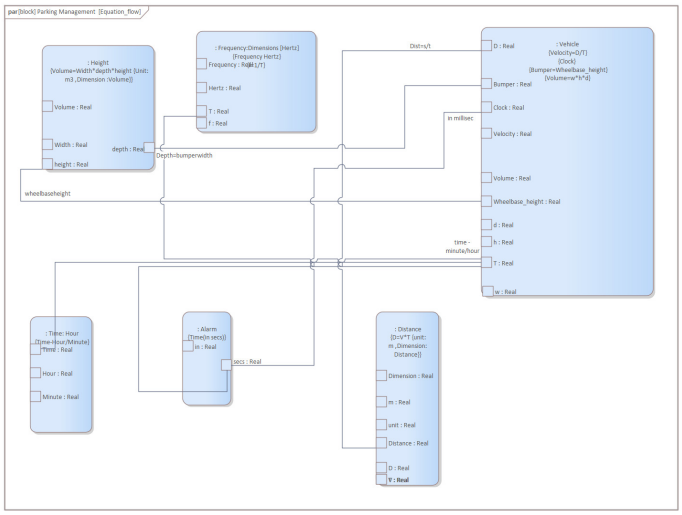

Fig. 6. parameters in parking system

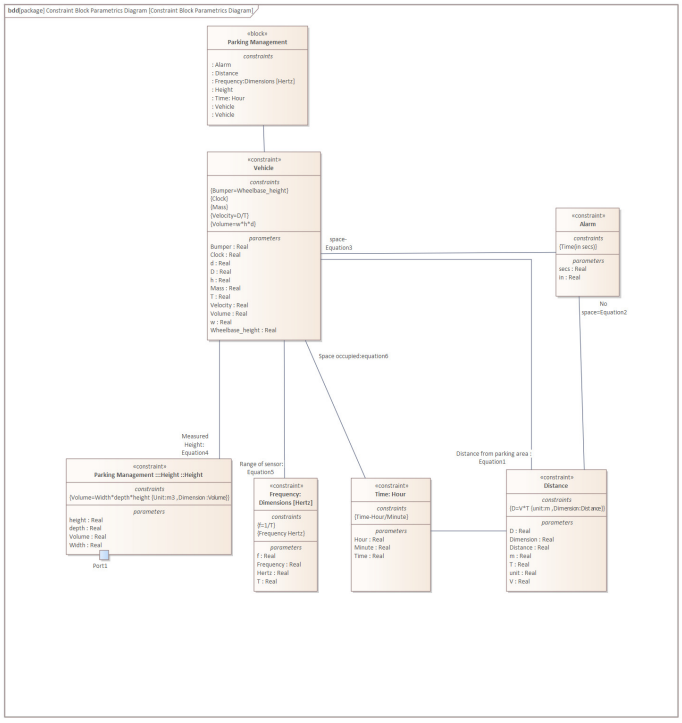

Fig. 7. Constraints for parking system

2) b.State Machine of accident and collision condition: In this Case vehicle is defined in different state and transit from one state to other. for example from below I can see the from a start state its moved to running state the vehicle will continuously be in running state until unless it will check any detection of trouble from checking state if any obstacle detected it will be going in sub-state of collision detection if the collision case has been cleared then it will move to final state have a glance on below figure.

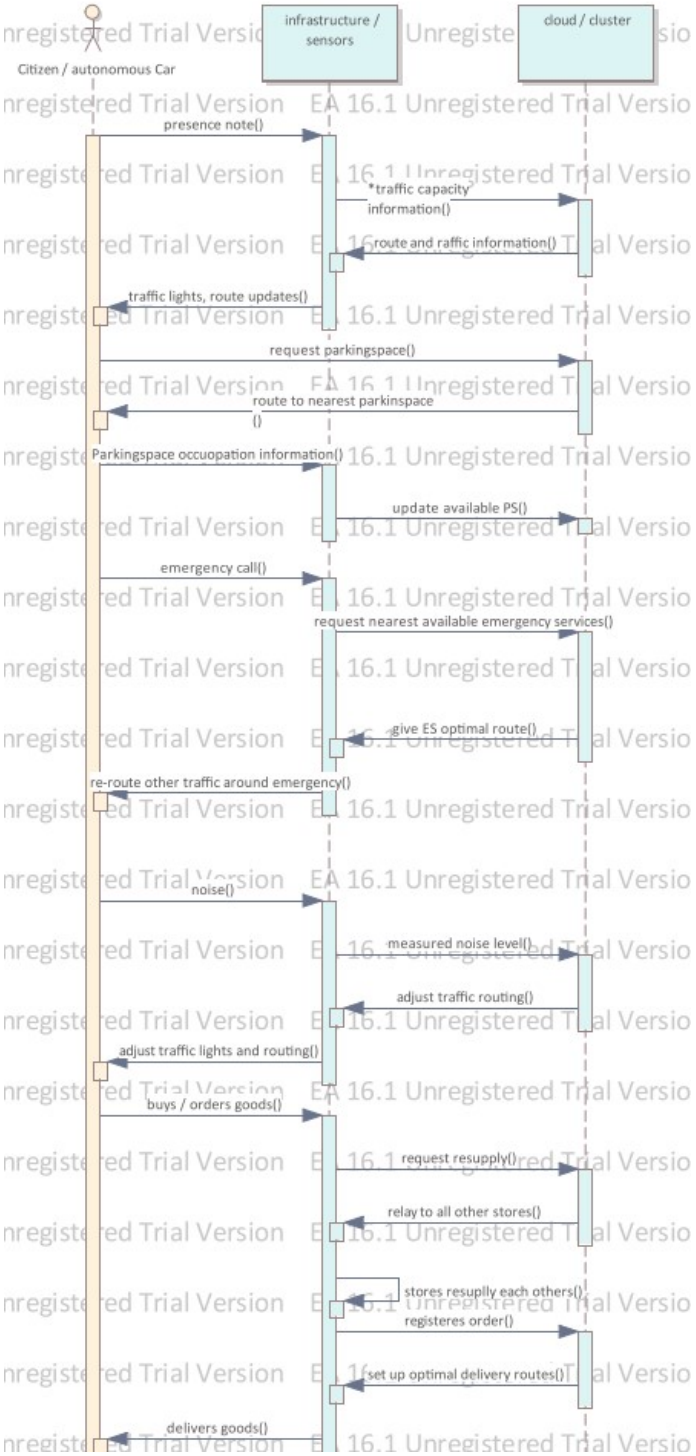

Fig. 8. Sequence diagram for different use-cases in a smart city

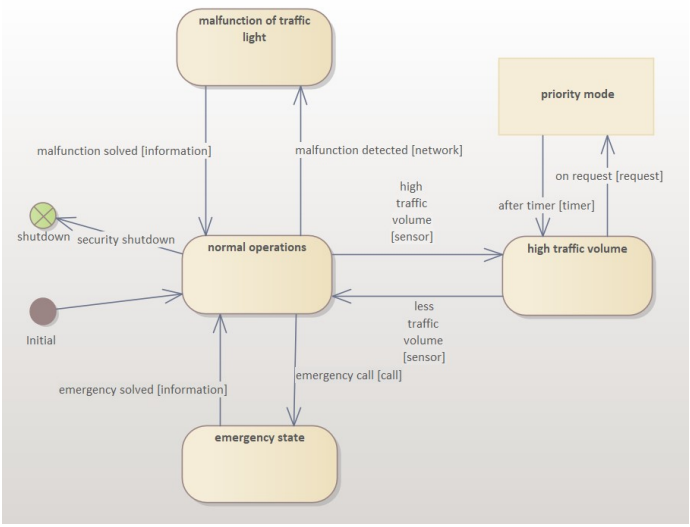

Fig. 9. State machine diagram for adaptive traffic control system

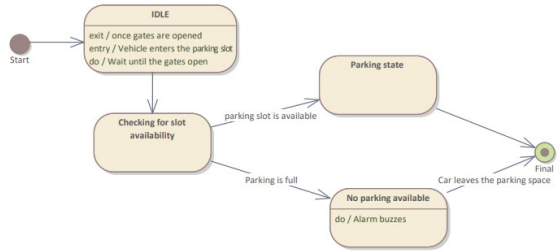

Fig. 10. State machine diagram for Parking system

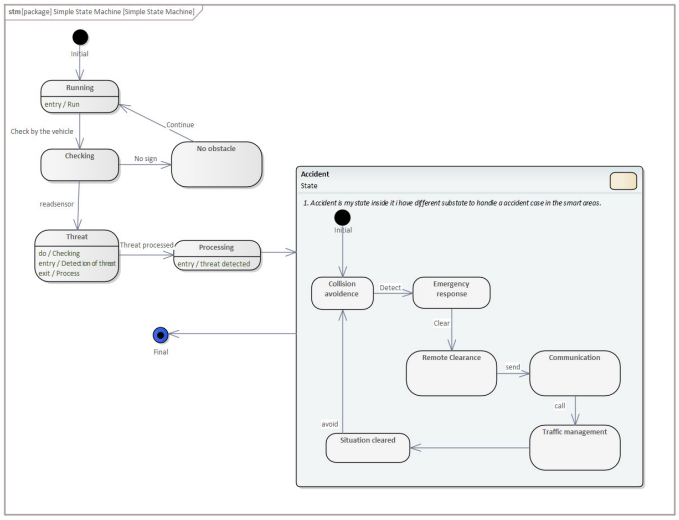

Fig. 11. Accident and Collision cases

Fig. 12.

IV. Component Implementation

A. Simulation in TINKERCAD

i) Description of components

Smart parking system comprises the Micro-controller unit(MCU), Ultrasonic sensor, 16x2 LCD, and piezo buzzer. Our prototype is a smart parking system that comprises only three spaces for testing. Fig. 13 consists of the circuit implemented in the TinkerCAD

1. **Micro-controller Unit (MCU)**

Arduino Uno, a microcontroller unit based on ATmega328P. It has 14 digital Input/output pins, six pins for PWM outputs. and 6 pins for analog input/outputs

2. **Ultrasonic sensor**

Parallax ping 3-pin ultrasonic sensor is used here. When influenced by the trigger pulse from the microcontroller unit, the sensor actuates a short burst of 40khz and is received by the sensor when it hits an obstacle. The pulse obtained is proportional to the distance between the sensor and the obstacle. Ultrasonic sensors are used for sensing the presence of vehicles in the parking spaces. This sensor is capable of measuring up to 200cm.

3. **LCD 16x2 with potentiometer**

LCD acts as an output interface for the car waiting at the entry. Here in this system 4-bit mode is being used in which the data is sent nibble by nibble with a higher nibble followed by the lower nibble. As this mode requires only 4 I/O pins. Potentiometer is connected with the LCD to vary the driving voltage which provides the feature of adjusting the contrast of the LCD when required.

4. **Piezo buzzer**

Piezo buzzer produces a sound that humans can hear when the voltage is supplied to the piezo crystal which changes the shape.

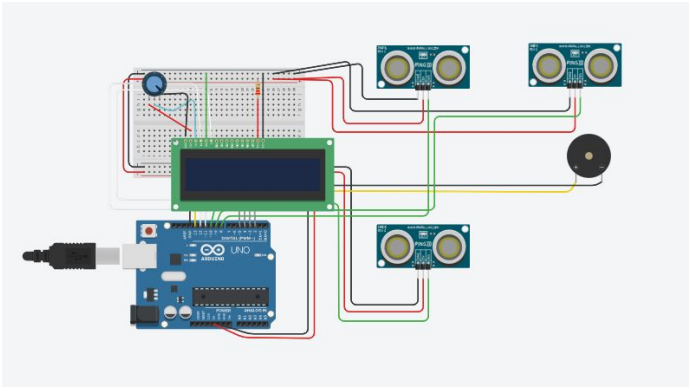

Fig. 13. Implementation in Tinkercad

```
void loop()
{
  float p1 = 0.01715* readDistance(t1, t1),p2 = 0.01715* readDistance(t2, t2),p3 = 0.01715* readDistance(t3, t3);

  Serial.println("Distance of the vehicle at parking space 1 -" +String(p1)+"cm");
  Serial.println("Distance of the vehicle at parking space 2 -" +String(p2)+"cm");
  Serial.println("Distance of the vehicle at parking space 3 -" +String(p3)+"cm");
```

Fig. 14. Variable declaration for distances measured by sensor

When no parking spaces are available then the buzzer will intimidate the car waiting at the entry with an alarm and also a message on the LCD.

Measuring distance using an ultrasonic sensor

Ultrasonic sensors could be placed in the ceiling in the case of covered parking, while in the case of open parking, they could be placed on the ground facing the car's platform. When the car is parked in the slot then the Ultrasonic sensor detects the presence. We have considered a threshold distance of 50 cm. If the distance between the ultrasonic sensor and the obstacle is below 50cm then it will be considered as occupied and the respective parking space's status will be changed in the display. For determining the distance between the obstacle. As seen in the Fig. 14.

Speed of sound – 343cm/s
Distance = (0.0343 cm/us * time taken by the sensor (us))/2

Here in this system, we have considered the threshold distance as 50cm. When the sensor reads a distance value below 50cm then it is occupied.

ii) Simulation Outputs

1) All spaces are available

The car waiting at the entry will be shared with information about the number of spaces available and which spaces are available. When spaces 1,2 and 3 are vacant then the ultrasonic sensor feeds the distance value to the MCU as greater than the threshold distance(Fig. 16). Then the LCD will display information that all spaces are available which is shown in Fig.15

2) Two parking spaces are available

In the given Fig. 17 one of the sensors reads a value of 44cm which is below the threshold distance so only two parking spaces will be available and which parking spaces will be determined by the logic in Fig. 18.

3) No parking space available

When all the sensors feed the distance value below the threshold distance then the status of the parking area changes to "No parking space available" which can be seen in Fig. 19 and buzzer alarms to the car waiting at the entry.

4) One parking space is available

In the case of the availability of only one parking space (which can be seen in Fig. 21) then which parking space it is could be determined by applying this logic in Fig. 22 i.e.

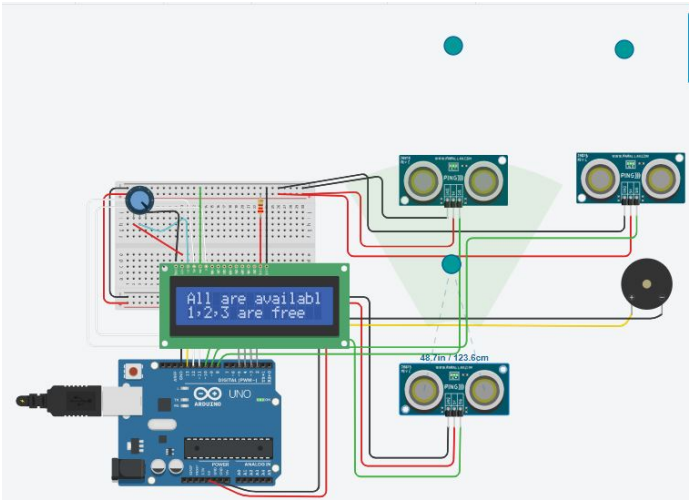

Fig. 15. All parking spaces are vacant

```
if(p1>50 & p2>50 & p3>50)
{
  lcd.setCursor(0,0);
  lcd.print("All are available  ");
  lcd.setCursor(0,1);
  lcd.print("1,2,3 are free    ");
  delay(500);
  endtime=micros();
  float executiontime;
  executiontime=endtime-start;
  Serial.print("execution time is ");
  Serial.print(executiontime);
  Serial.print("microseconds\n\n");
  start=endtime;
      }
```

Fig. 16. Condition for the availability of all parking spaces

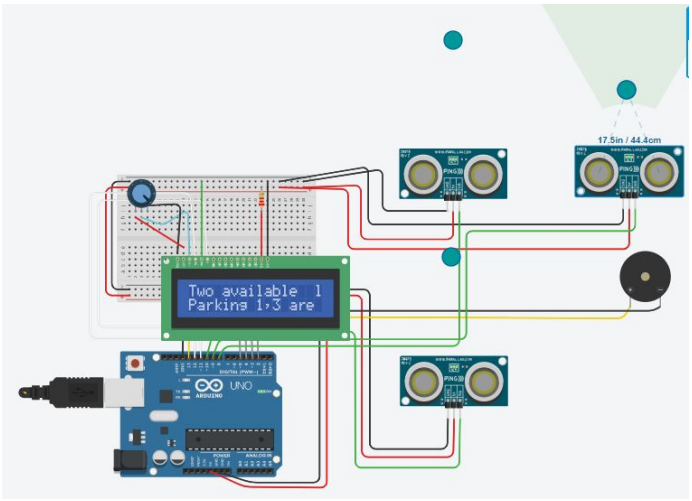

Fig. 17. Two parking spaces are vacant

```
else if((p1>50 & p2>50)|(p2>50 & p3>50 )|( p3>50 & p1>50))
{
  lcd.setCursor(0,0);
  lcd.print("Two available  ");
  lcd.setCursor(0,1);
  if(p1>50 & p2>50)
```

Fig. 18. Finding the number of the two available spaces

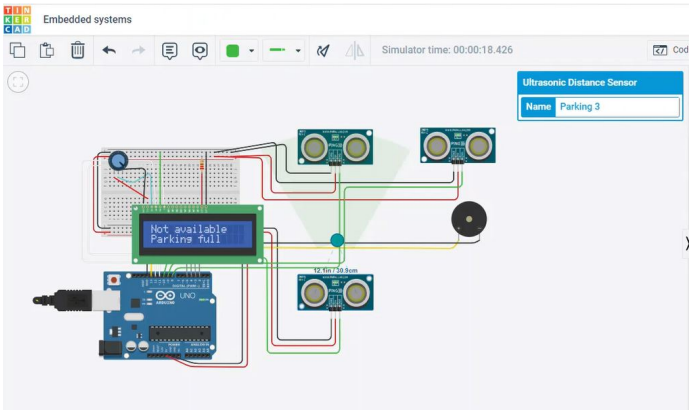

Fig. 19. No parking spaces are vacant

by comparing whichever is greater than the threshold distance.

iii) Worst Case Execution Time calculation (WCET)
In this system, we are using the micros() function for calculating WCET for the four tasks. We have declared the start time during initialization in the void setup() i.e. in this system when the MCU triggers the ultrasonic sensor. And have declared endtime at each task.

The execution time for each task will be the difference between the start and endtime. Then the start will further be swapped with the present endtime value for the calculation of the next task's execution time.

Among the four tasks, the worst case execution time(WCET) occurred for the task which intimidates with no space available and alarming is 1.83 seconds. This could be due to the inclusion of a buzzer alarm which delays the completion of the task and the average worst-case execution time calculated for this task.

```
    else if ((p1<50 & p2<50 & p3<50))
  {
    lcd.setCursor(0,0);
    lcd.print("Not available    ");
    lcd.setCursor(0,1);
    lcd.print("Parking full     ");
    delay(500);
    tone(Buzzer, 500);
  delay(1000);
  noTone(Buzzer);
  delay(100);
endtime=micros();
```

Fig. 20. Buzzer alarm function

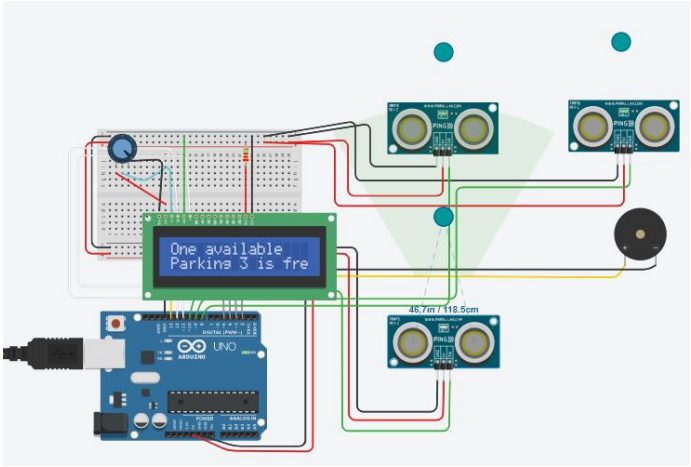

Fig. 21. One parking space

```
else if((p1<50 & p2<50)|(p2<50 & p3<50)|(p1<50 & p3<50))
 {
 lcd.setCursor(0,0);
 lcd.print("One available   ");
```

Fig. 22. Finding which one parking space is available

B. Time Scheduling

After an estimated result from Hardware Simulation using Tinker-cad we have provided a short interface of hardware to software also check whether the system is schedules.
NOTE: we have used an Arduino board in our simulation which doesn't provide an accurate result to a real-time system as timing generally comes in milliseconds.

1) a.Using PYCPA: It is a Python implementation of Computational Performance Analysis and method or classes which use python programming to calculate a result analysis on the basis of event model ,Gantt chart have a look below here

- we assigned one resource two different tasks (mainly

```
{
lcd.begin(16,2);
  lcd.setCursor(0,0);
  Serial.begin(9600);
  start= micros();
    }
```

Fig. 23. Micros function initialization

```
    endtime=micros();
  long int executiontime;
  executiontime=endtime-start;
  Serial.print("execution time is ");
  Serial.print(executiontime);
  Serial.print("microseconds\n\n");
start=endtime;
}
```

Fig. 24. Execution time calculation

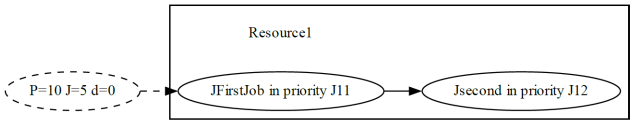

Fig. 25. Tasks taken in one Resource

```
##now define the resources
r1 = s.bind_resource(model.Resource("Resource1", schedulers.SPPScheduler()))
#r2 = s.bind_resource(model.Resource("Resource2", schedulers.SPPScheduler()))

##create and bind Jobs to resource1
J11 = r1.bind_task(model.Task("JFirstJob in priority J11", wcet=1,  bcet=0.5, scheduling_parameter=1))
J12 = r1.bind_task(model.Task("Jsecond in priority J12", wcet=1.83,  bcet=0.5,scheduling_parameter=2))
#create and bind jobs to resource2
# J21 = r2.bind_task(model.Task("JFirst in priority J21 ", wcet=1,  bcet=0.5, scheduling_parameter=3))
# J22 = r2.bind_task(model.Task("JSecond in priority J22", wcet=1.83,  bcet=1,scheduling_parameter=4))

# specify precedence constraints: J11 -> J21; J12-> J22
J11.link_dependent_task(J12)
# J12.link_dependent_task(J22)

# register a periodic with jitter event model for J11 and J12
#here P is the period , jitter -computation gap
J11.in_event_model = model.PJdEventModel(P=10, J=5)
J12.in_event_model = model.PJdEventModel(P=20, J=6)

# plot the system graph to visualize the architecture
g = graph.graph_system(s, filename='Tasks.pdf',dotout='Tasks.dot')

# perform the analysis
print("Performing analysis")
Plot_job = [J11,J12]
task_results = analysis.analyze_system(s)
# print the worst case response times (WCRTs)
sim = simulation.SimTask('q',J11)
```

Fig. 26. System Schedulability check

the task in our case is one providing an output to LCD screen as space available or not available for allocation.

- Alarm beep sound if no space can be defined so by doing these we can overcome the vehicle time which are in queue.

C. Equations

From previous analysis with hardware simulation the worst case execution time(wcet)are around in milliseconds such as 1 and 1,83 so we have defined two periods(T1,T2) and delay between the tasks(as jitter) the calculated results are below. d= distance s= speed t=time wcet: worst case execution time Ci=computational time Ti=period of task Ui= utilization time whereas i=1....n
C11= Computational time for task 1, C12 = Computational time for task 2, T11,T12= Time period.

In our case:
C11=1 T11=10 C12=1.83 T12=10 Here Ui¡1 as we get 0.8356

$$d = s * t$$

Ui= $\sigma(ci/Ti)$
(1)

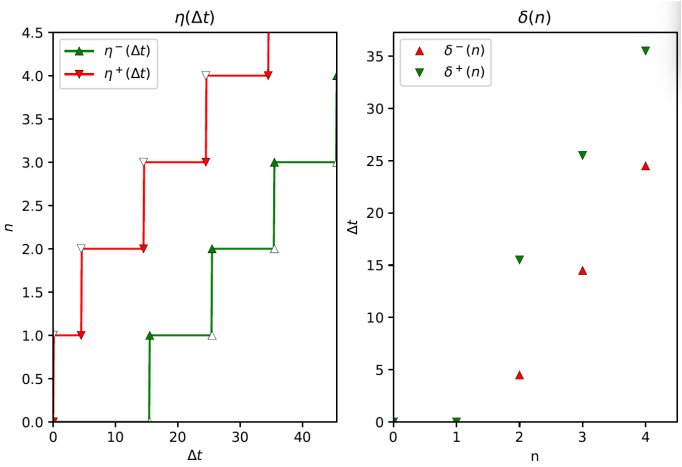

Fig. 27. Event model of T12

V. Implementation of Parking System

The parking system use case is implemented in C++. The implementation is aligned with the state machine diagram and is implemented in a organised approach where we have classes such as Parking, Vehicle and functions to check slot availability and to open gates for the parking lot. We have considered area available in slot and making a decision whether slot is empty or not.

A. State Machine Implementation

We have used a even-driven approach in this implementation with the help of switch statement to transit between each states. Vehicle will be in Idle state initially and as gates open we will transit to check slot availability state. once slots are available we will transit to parking state and exit state if not available or parking is done.

B. Functions

In our implementation we have different functions such as Handlestates, Check for slot availability and Opengates function. More details can be witnessed in our github project and in our code snippets too.

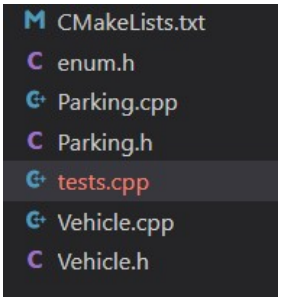

Fig. 28. File structure of code

```
void Vehicle::HandleStates(int p1,int p2,int p3,bool gateopen)
{
    CParking* parkingobj=new CParking;
    VehicleState nextState=VehicleState::Idle; // Assuming Vehicle is in IDLE state
    for(int i=0;i<10;i++)
    {
        switch(nextState)
        {
            case VehicleState::Idle:
            /// cout << "entered Vehicle State IDLE" << endl;

                if(parkingobj->OpenGates(gateopen)==true)
                {
                    nextState = VehicleState::CheckingState;
                }
                break;

            case VehicleState::CheckingState:
                ///cout << "Checking slot" << endl;
                nextState =parkingobj->CheckingSlotAvailability(p1,p2,p3);
                break;

            case VehicleState::Parking:
                nextState= VehicleState::exit;
                break;

            case VehicleState::NoParking:
                nextState= VehicleState::exit;
                break;

            default :
                cout << "final State" << endl;
                break;
        }
    }
}
```

Fig. 29. Parking system implementation code snippet

```
VehicleState CParking::CheckingSlotAvailability(int p1,int p2,int p3)
{
    VehicleState nextState;
    ///cout << "function call" << endl;
    if(p1>50 & p2>50 & p3>50)
    {
        cout << "All slots are available "<< endl;

        nextState=VehicleState::Parking;
    }
    else if((p1>50 & p2>50)||(p2>50 & p3>50 )||( p3>50 & p1>50))
    {
        cout << "Two slots are available" << endl;
        if(p1>50 & p2>50)
        cout << "Parking 1,2 are free" << endl;
        else if(p2>50 & p3>50)
        cout << "Parking 2,3 are free" << endl;
        else if(p3>50 & p1>50)
        cout << "Parking 2,3 are free" << endl;

        nextState=VehicleState::Parking;
    }
    else if ((p1<50 & p2<50 & p3<50))
    {
        cout << "Parking full " << endl;

        nextState=VehicleState::NoParking;
    }
```

Fig. 30. Function implementation

VI. Conclusion

From above calculation, we receive utilization below so the implemented component is ”Schedulable”from EDF(End Deadline First)

A. Output from PYCPA

Analysis of Result:

- JFirstJob in priority J11: wcrt=1
  bwcrt=q*WCET:1*1=1
  Jsecond in priority J12: wcrt=2
  bwcrt=JFirstJob in priority J11:eta*WCET:1*1=1,
  q*WCET:1*1.83=1.83

with the above performanace with execution time and block time from other tasks a valid result obtained here.

B. Graph Analysis

The graph shown for both End Deadline First(EDF) which deals with deadline and the Rate Monotonic system (RMS) by which period is main concern.

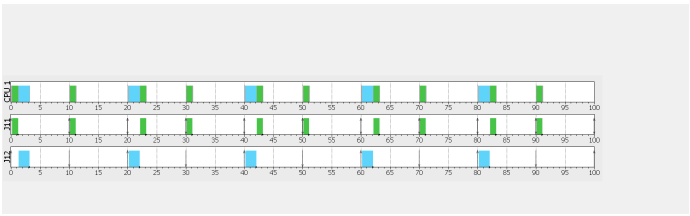

Fig. 31. EDF analysis

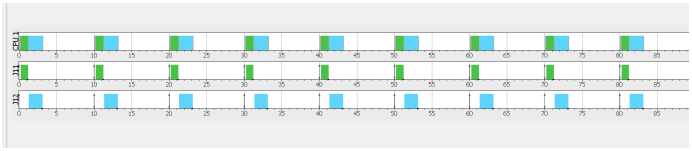

Fig. 32. Rate monotonic system

C. Inspection

Review of Hardware prototype simulation.

Improving points:

- 1.Hardware simulation with tinker Cad Code is not implemented with state machine of parking system it performed action in if else condition

  2.Two task implementation has been occupied in one function overall the compilation of one function is in account.

Positive response

- 1.The execution time has been rectified by using start and end time in all if /else statement however after the first task execute and so on. After all the if else the added the time for one execution.

  2.The state machine implementation inside code which is being missed in Hardware simulation code has been reviewed and rectified in the code of the parking system which has been performed in visual studio system code by using switch cases inside it.
  Different files has been created such as : vehicle ,Parking etc everything covered in testing part.

D. Testing

This section covers the testing of implemented code. Testing at this stage helps in reducing the defects costs at later stages. This is divided into unit tests, defects tests and components tests.

E. Unit testing

For unit testing we have used google test framework since the implementation is in C++. This part majorly covers the testing in the form of units which is function in this case. Every functions output value is checked against the expected output.

```
TEST(CheckingSlotAvailability,positive3 )
{
    VehicleState previousState;
    const char* str1;
    str1 = vehicleobj.VehicleStatestr(VehicleState::Parking);
    previousState= parkingobj.CheckingSlotAvailability(40,40,100);
    const char* str2= vehicleobj.VehicleStatestr(previousState);
    ASSERT_STREQ(str1,str2);

    previousState= parkingobj.CheckingSlotAvailability(100,40,100);
    const char* str3= vehicleobj.VehicleStatestr(previousState);
    ASSERT_STREQ(str1,str3);

    previousState= parkingobj.CheckingSlotAvailability(100,40,100);
    const char* str4= vehicleobj.VehicleStatestr(previousState);
    ASSERT_STREQ(str1,str4);
}
```

Fig. 33. Unit Testing

```
TEST(CheckingSlotAvailability,negative1 )
{
    VehicleState previousState;
    const char* str1;
    str1 = vehicleobj.VehicleStatestr(VehicleState::NoParking);
    previousState= parkingobj.CheckingSlotAvailability(40,40,40);
    const char* str2= vehicleobj.VehicleStatestr(previousState);
    ASSERT_STREQ(str1,str2);

    previousState= parkingobj.CheckingSlotAvailability(-1,40,40);
    const char* str4= vehicleobj.VehicleStatestr(previousState);
    ASSERT_STREQ(str1,str4);

    previousState= parkingobj.CheckingSlotAvailability(-1,-1,-1);
    const char* str3= vehicleobj.VehicleStatestr(previousState);
    ASSERT_STREQ(str1,str3);
}
```

Fig. 34. Defect Tests during implementation

F. Defect Tests

Defect tests can be done at implementation level and also during the validation stage. Below snippets are those which shows these tests at code level as negative inputs and the test cases which can be tested as part of validation.

G. Component Tests

In this we have covered component tests by checking the data transfer between two classes since all we have is simulation environment

	S.No	Test Case	Test Steps	Result
2	TC_1	Validate if Vehicle is not able to park when all the slots are full	1. Modify sensor values such that all slots are full	PASS
3	TC_2	Validate if Vehicle is not able to park when all the slots are empty	1. Modify sensor values such that all slots are empty	PASS
4	TC_3	Validate if we are able to open the gates if they are closed	1. Send the value as gate is closed	PASS
5	TC_4	Validate if we are able to park if one slot is empty	1. Modify sensor values such that one slot is available	PASS
6	TC_5	Validate if we are able to park if two slots are empty	1. Modify sensor values such that two slots are available	PASS

Fig. 35. Defect Tests

VII. Label Section

- 1. Sebastian Rudolph
  a) Activity diagram for traffic control system
  b) Sequence diagram for smart city
  c) State diagram for emergency services traffic control
  d) creation of Powerpoint presentation

  2. Vasanth Kandiyathevan
  Implementation of smart parking system in Tinkercad.
  a) Components finalization and description
  b) Simulation in Tinkercad
  c) Worst-Case Execution Time calculation

  3. Divya Bharti Sharma:
  Covered Design analysis and implementation part are:
  a) Introduction
  b) Requirements
  c) Automotive domain characteristics.
  d) Activity diagram for Detection at remote areas.
  e) Block definition diagram including internal block diagram for the requirements.
  f) Parametric diagram and equation constrained for parking system.
  g) State machine implementation with respect to accident and collision scenario as remaining covered by fellow colleague.
  h) Scheduling from PYCPA and SIMSO.
  i) Inspection(Review of Hardware simulation with positiev and negative impacts)
  j) Acknowledgement
  h) References

  4. Jahnavi Vennapusa

  For parking system below tasks are carried out:
  a) Activity Diagram
  b) State Machine diagram
  c) Implementation with C++ to align with state machine diagram
  d) Unit testing with Google Test FrameWork
  e) Defect tests
  f) Component or interface tests

Acknowledgment

We gratefully acknowledge [Profeessor Stefan Henkler] for assistance with [Traffic Control Smart City, approach],Towards a new phase of Technology in Embedded world. I would also like to thank (Sebastian Rudolph,master Student,Fachhochschule Dortmund),(Jahnavi Vennupusa,Master Student, Fachhochschule Dortmund),(Vasanth Kandiyathevan ,Master Student,Fachhochschule Dortmund) for the continuous alignment with projects and significant ideas.

[2] [6] [4] [6] [5] [**?**] [1] [3]

REFERENCES

[1] J Cynthia, C Bharathi Priya, and PA Gopinath. Iot based smart parking management system. *International Journal of Recent Technology and Engineering (IJRTE)*, 7(4S):374–379, 2018.

[2] Mircea Eremia, Lucian Toma, and Mihai Sanduleac. The smart city concept in the 21st century. *Procedia Engineering*, 181:12–19, 2017.

[3] Yanfeng Geng and Christos G Cassandras. A new "smart parking" system infrastructure and implementation. *Procedia-Social and Behavioral Sciences*, 54:1278–1287, 2012.

[4] Khaoula Hassoune, Wafaa Dachry, Fouad Moutaouakkil, and Hicham Medromi. Smart parking systems: A survey. In *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pages 1–6. IEEE, 2016.

[5] Mamata Rath and Bibudhendu Pati. Communication improvement and traffic control based on v2i in smart city framework. *International Journal of Vehicular Telematics and Infotainment Systems (IJVTIS)*, 2(1):18–33, 2018.

[6] M Mazhar Rathore, Anand Paul, Seungmin Rho, Murad Khan, S Vimal, and Syed Attique Shah. Smart traffic control: Identifying driving-violations using fog devices with vehicular cameras in smart cities. *Sustainable Cities and Society*, 71:102986, 2021.